Project 1 IT3105:

Implement and Compare the Performance of AI Algorithms to Solve The n-Queens Problem

Lab Goals

- Implement a recursive algorithm to solve the 8-queens problem.
- Upgrade your recursive algorithm to handle more generic cases (n-queens).
- Implementation and comparison of heuristic algorithms for the n-queens problem.

Groups Allowed? You may work alone or in groups of two.

Deadline: September 26, 2016 (Monday) at $11:45$ PM.

Assignment Details

You will implement and use several AI algorithms to solve the n-queens problem. The n-queens problem is the problem of placing n chess queens on an $n \times n$ chessboard so that no two queens threaten each other. this project has three parts. In the first part, you need to solve the 8-queens problem using a recursive algorithm strategy called *backtracking*. The 8-queens problem is a particular example of the more general n-queens problem. In this version, you have to place 8 non-attacking queens on a 8×8 chessboard. For the second part, you need to upgrade your recursive algorithm to handle more generic cases of (n-queens). For these two parts, you need to implement an interactive input method which is described in the later part of this project description.

For the last and third part of this project, you need to implement 3 different AI algorithms to solve the n-queens problem. The algorithms are: (*i*) tabu search (TS), (*ii*) simulated annealing (SA), and (*iii*) genetic algorithm (GA). You also need to compare the performances of these three algorithms.

The n-Queens Problem

Suppose that we are given an $n \times n$ chessboard, where n is a positive integer. In the game of Chess a queen can move any number of spaces in any linear direction: horizontally, vertically, or along a diagonal (Fig. 1). As such, a queen can attack in horizontal, vertical, and the two diagonal directions. The n-queens problem asks to find a configuration of n queens on an $n \times n$ chessboard such that no two attack one another. Apparently, any solution to the n-queens

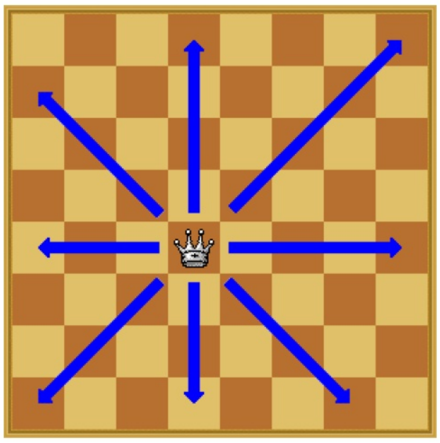
Figure 1: Permitted movement for a queen in a chessboard.

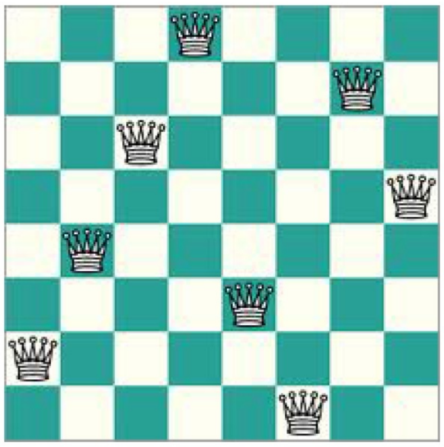
Figure 2: A solution to the 8-queens problem.

problem has exactly one queen in each row. There are no solutions to this problem for $n = 2$ or 3, but a solution exists for each $n \geq 4$ [1]. For instance, Fig. 2 pictures one solution to the traditional 8-queens problem.

Algorithms

All the algorithms necessary for this project (recursive algorithm with backtracking, tabu search (TS), simulated annealing (SA), and genetic algorithm (GA)) will be discussed in lectures.

Things To Do

Part: 1

For the first part, the assignment is to write a recursive program that implements backtracking to solve the 8-queens problem. The accepted solutions are board arrangements, where there are no threats between the queens. Your code has to allow user input. The input allows user to specify the positions for the first n queens, where $0 < n < 8$, in a format describing the positions of the queens on the board (column-wise) with a one dimensional array. For example, using this format the solution in Fig. 2 are represented as:

>> **2 4 6 8 3 1 7 5**

There are many convenient ways to encode the board. **In developing your program to solve this project, you are allowed to encode the board in your own ways.** However, for the demo, you must follow the earlier mentioned way as input. If the user specifies the positions of interest for the first $n > 0$ queens, your program should start looking for solutions starting from the column $n + 1$. If there is already a threat between the queens with the input supplied by the user, the program reports that a solution is impossible for the specified input. Otherwise, the program look for possible solutions starting by placing the queens in the $n+1$ and subsequent columns and report any complete solution which matches the user's specification.

For example, the user would like to give the input in a way in which the first three columns are taken by the queens placed on the second, fourth, and sixth rows, the user will do the following:

```
>> 2   4   6   0   0   0   0   0
```

In this case, your program shall find any legal solution for the remaining five columns. Hence the positions specified with zero "0" in the user input denote the empty columns those your program will find to solve the 8-queens problem. For output, your program should print the complete legal solution according to the format mentioned earlier.

Part: 2

For the send part, you need to upgrade your recursive algorithm to handle n-queens problem, where n can be any value and must be $n \geq 4$. Also, your program shall find **all complete legal solutions (if any)** for the remaining columns for a given input. Note that there exists more than one legal solutions for n-queens problem, where $n \geq 4$. The input method will be as earlier, except the inclusion of the value for n, at first. for example, for the input:

```
>> 8
>> 1   7   0   0   0   0   0   0
```

the output will be **all the complete legal solutions** (if any) :

```
>> 1   7   4   6   8   2   5   3
>> 1   7   5   8   2   4   6   3
>> 1   7   _   _   _   _   _   _
>> :   :
>> :   :
>> :   :
```

Part: 3

For this part, you assignment is to implement 3 AI algorithms to solve the n-queens problem using a complete input initial positions for the board. The algorithms are: (*i*) tabu search (TS), (*ii*) simulated annealing (SA), and (*iii*) genetic algorithm (GA). Note that the initial complete board positions may contain queens at the attacking positions. And, you need to use the same input for all the three algorithms.

The input will contain the size of the board, and a initial complete initial board positions. for example:

```
>> 8
>> 1   7   4   6   3   3   4   2
```

The output will be **all the complete legal solutions** (if any) like the one mentioned in Part: 2.

You also need to compare the performances of these three algorithms based on any specific initial board positions and the board size n, which will be supplied during the demo. For this you need to record and show the following data during the demo.

Board Size (n)	No. of Soln for TS	Time to Calculate for TS	No. of Soln for SA	Time to Calculate for SA	No. of Soln for GA	Time to Calculate for GA
–	–	–	–	–	–	–

For all the three parts, there are reserve points for showing the final complete legal solution step by step based on the input. This is not mandatory. However, if you don't show the step by step, you will loose the reserve points.

Demo

There will be a demo session where you will show us the running code and we will verify that it works. This demonstration can give a maximum of 25 points. As mentioned earlier, 5 ($1+1+3$, for Part 1, 2, 3, respectively) of these 25 points will be reserved for showing the final complete legal solution step by step from the input board positions. Whenever you will find more than one complete legal solutions, it is enough to show the step by step procedure for just one solution.

Report

You should write a report answering the points below. The report can give a maximum score of 15 points. Your report must not exceed 4 pages in total. **Over length reports will result in points being deducted from your final score.** Print on both sides of the sheets, preferably. **Bring a hard copy of your report to the demo session.**

a) Part–1 (2p)

- Describe your chosen board representation, and the reason behind your choice. (1p)
- Describe your heuristic function. (1p)

b) Part–2 (2p)

- For a fixed input (your choice, but for the first four columns only), how many solutions your program can find for $n = 14$, 16, 18, and 20. (1p)
- For a the same input, the time taken by your program to find all the solutions for $n = 14$, 16, 18, and 20. (1p)

c) Part–3 (11p)

- You will use the same inputs for TS, SA, and GA. Mention your fixed inputs for $n = 18$, 20, 24, and 30. (1p)
- For TS. (1.5p)
  - Describe your implementation. (1p)
  - Describe your fitness function. (0.5p)
- For SA. (1.5p)
  - Describe your implementation. (1p)

  - Describe your fitness function. (0.5p)

- For GA. (5p)
  - Describe your fitness function. (0.5p)
  - A detailed description of the complete GA design. This includes the chromosome representation, implementation of crossover and mutation operators, and your selection strategy. (2.5p)
  - Describe whether or not the crossover and mutation operators will produce infeasible off-spring(s) after executing. If yes, how did you handle that. If not, why? (1p)
  - A table summarizing values of the following parameters which gave you the best result: (1p)

    {population size, number of generations, crossover rate, mutation rate}

- Comparison. (2p)
  - For the fixed inputs used in this part for $n = 18$, 20, 24, and 30; give the comparison following the table format mentioned earlier.

Delivery

You should deliver your report + a zip file of your code on *itslearning*. The submission system will be closed at 11 : 45 PM on September 26, 2016. The 40 points total for this project is 40 of the 100 points available for this class.

For this project you can work alone or in groups of two. If you work in a group, both memebers of the group need to deliver on *itslearning*. Both group members must be registered as part of the submission on *itslearning*. Also, **both group members need to attend the demo session.**

References

[1] E. Hoffman, J. Loessi, and R. Moore, "Constructions for the solution of the m queens problem," *Mathematics Magazine*, vol. 42, no. 2, pp. 66–72, 1969.